Attorney's Docket No.: 10559-641001 / P12484

APPLICATION

FOR

UNITED STATES LETTERS PATENT

TITLE: RETURN ADDRESS STACK

APPLICANT: STEPHAN J. JOURDAN, JOHN ALAN MILLER AND NAMRATHA JAISIMHA

CERTIFICATE OF MAILING BY EXPRESS MAIL

Express Mail Label No EL870691273

I hereby certify that this correspondence is being deposited with the United States Postal Service as Express Mail Post Office to Addressee with sufficient postage on the date indicated below and is addressed to the Commissioner for Patents, Washington, D C. 20231.

December 21, 2001
Date of Deposit

Signature

Gabe Lewis
Typed or Printed Name of Person Signing Certificate

RETURN ADDRESS STACK

BACKGROUND

This invention relates to storing and retrieving return addresses.

A system architecture describes the mode of operation of a processor and mechanisms to support operating systems and executives, including system-oriented registers and data structures and system-oriented instructions. The system architecture also provides support necessary for switching between real-address and protected modes.

Execution of instructions in a pipelined processor often requires predicting the path of execution before the results of branch instructions may be known. A return stack buffer (RSB) is often used to store predicted return addresses for subroutine CALL and RETURN instructions.

DESCRIPTION OF DRAWINGS

FIG. 1 is a block diagram of a processor.

FIG. 2A is a block diagram of an first embodiment of a return address stack.

FIGS. 2B-2D are flowcharts showing a process for storing and retrieving return addresses using the stack of FIG. 2A.

FIG. 3A is a block diagram of a second embodiment of a return address stack.

FIGS. 3B-3C are flowcharts showing a process for storing and retrieving return addresses using the stack of FIG. 3A.

DETAILED DESCRIPTION

Referring to FIG. 1 a processor 10 is shown. The processor 10 is a super-scalar, pipelined architecture. The term "super-scalar" means that, using parallel processing techniques, the processor 10 may decode, dispatch, and complete execution (retire) of multiple instructions per clock cycle (on average). To handle this level of instruction throughput, the processor 10 uses a decoupled, multiple stage pipeline that supports out of order instruction execution. The micro architecture pipeline of the processor 10 is divided into several sections, i.e., a first level cache 12 and a second level cache 14, a front end 16, an out of order execution core 18, and a retirement section 20. Instructions and data are supplied to these units through a bus interface unit 22 that interfaces with a system bus 24. The front end 16 supplies instructions in program order to the out of order core 18 that has very high execution bandwidth. The front end

16 fetches and decodes instructions, and breaks the instructions down into simple operations called micro-ops (µ-ops). The front end 16 can issue multiple µ-ops per cycle, in original program order, to the out of order core 18. The front end 16 performs several basic functions. For example, the front end 16 performs prefetching of instructions that are likely to be executed, fetching instructions that have not already been prefetched, decoding instructions into micro operations, generating micro code for complex instructions and special purpose code, delivering decoded instructions from an execution trace cache 26, and predicting branches using advanced algorithms in a branch prediction unit 28. The front end 16 of the processor 10 minimizes the time to decode instructions fetched from the target and minimizes wasted decode bandwidth due to branches or branch target in the middle of cache lines.

The execution trace cache 26 stores decoded instructions. Instructions are fetched and decoded by a translation engine (not shown) and built into sequences of µ-ops called traces. These traces of µ-ops are stored in the trace cache 26. The instructions from the most likely target of a branch immediately follow the branch without regard for continuity of instruction addresses. Once a trace is built, the trace cache 26 is searched for the instruction that follows that trace.

If that instruction appears as the first instruction in an existing trace, the fetch and decode of instructions 30 from the memory hierarchy ceases and the trace cache 26 becomes the new source of instructions.

The execution trace cache 18 and the translation engine (not shown) have cooperating branch prediction hardware. Branch targets are predicted based on their linear addresses using Branch Target Buffers (BTBS) 28 and fetched as soon as possible. The branch targets are fetched from the trace cache 26 if they are indeed cached there; otherwise, they are fetched from memory e.g., cache or main memory. The translation engine's branch prediction information is used to form traces along the most likely paths.

The execution core 18 executes instructions out of order, which enables the processor 10 to reorder instructions so that if one µ-op is delayed while waiting for data or a contended execution resource, other µ-ops that are later in program order may proceed around it. The processor 10 employs several buffers to smooth the flow of µ-ops. This implies that when one portion of the pipeline experiences a delay that delay may be covered by other operations executing in parallel or by the execution of µ-ops which were previously queued up in a buffer.

The core 18 is designed to facilitate parallel execution. The core 18 can dispatch multiple µ-ops per cycle. Most pipelines can start executing a new µ-op every cycle, so that several instructions can be processed any time for each pipeline.

The retirement section 20 receives the results of the executed µ-ops from the execution core 18 and processes the results so that the proper architectural state is updated according to the original program order. For semantically correct execution, the results of instructions are committed in original program order before it is retired. Exceptions may be raised as instructions are retired. Thus, exceptions cannot occur speculatively. They occur in the correct order, and the processor 10 can be correctly restarted after execution.

When a µ-op completes and writes its result to the destination, it is retired. A ReOrder Buffer (ROB) (not shown) in the retirement section 20 is the unit in the processor 10 which buffers completed µ-ops, updates the architectural state in order, and manages the ordering of exceptions.

The retirement section 20 also keeps track of branches and sends updated branch target information to the BTB 28 to update branch history. In this manner, traces that are no longer needed can be purged from the trace cache 26 and new branch paths can be fetched, based on updated branch history information.

In a pipelined processor, such as processor 10, instructions are "speculatively" fetched by front end 16 from first level cache 12 and second level cache 14. Speculatively fetching instructions refers to predicting the path of execution of a set of instructions being executed in a pipeline, that is, having to predict a path of fetched instructions before the decoding of those instruction can be completed. Predicting a path of execution includes predicting a RETURN address that may follow a sub-routine CALL instruction. A sub-routine CALL instruction is sometimes referred to as an "unconditional branch" instruction.

In the case of unconditional branch instructions, such as subroutine CALLs, a return stack buffer (RSB) is often used to store predicted RETURN addresses that may follow the CALL instruction. Using a traditional RSB, predicted RETURN addresses are pushed onto the RSB for each CALL instruction fetched, and the predicted RETURN address is popped from the stack when a RETURN instruction is predicted. A pointer to the top of the RSB stack (TOS) is incremented with each push of an address onto RSB, and decremented with each pop of an address from RSB. In some cases, when speculatively fetching instructions, the traditional RSB structure fails to preserve a predicted RETURN address. For example, a first CALL instruction (CALL1) is fetched, its predicted return address (RETURN1) is pushed onto the stack (often, the predicted return address is the address immediately after the address of the CALL instruction). After RETURN1 is pushed onto RSB, TOS is incremented to point to the next entry in RSB. When a return instruction is fetched, TOS is decremented to point to the stack location holding RETURN1 address and RETURN1 is popped from the stack as the predicted return address. If a second CALL instruction (CALL2) is fetched, pushing its predicted RETURN address (RETURN2) onto RSB, RETURN1 address will be overwritten with RETURN2 address in the RSB (TOS having remained the same). If the predicted path of execution for the return instruction is not correct (for example if an intervening branch instruction result is mis-predicted), RETURN1 address will be lost (it is no longer on RSB, having been over-written with RETURN2 address). Therefore, the next return instruction fetched will be mis-predicted as having a RETURN2 address instead of the RETURN1 address.

Referring to FIG. 2A, in a first embodiment, processor 10 includes a two part return address buffer 40, which can be part of the the Branch Target Buffers (BTBS) 28 (Fig. 1). The return address buffer 40 stores predicted RETURN addresses and allows recovery of predicted RETURN addresses in the case of mis-predicted instruction fetching (as described above). Address buffer 40 includes a Speculative RSB 42 (SRSB 42) and a Committed RSB (CRSB 44), both of which having multiple entries that may include predicted return addresses that have been pushed onto buffer 40 by front end 16. When a predicted return address stored in buffer 40 is popped by front-end 16, the predicted return address may come from either SRSB 42 or CRSB 44.

SRSB 42 entries are up-dated when front-end 16 fetches a new CALL instruction and pushes a new predicted return address onto buffer 40 (typically, the predicted return address is the next instruction after the CALL instruction). If all of the SRSB 42 entries are full, an SRSB entry will be over-written with the new return address. Whenever an SRSB 42 entry is determined to have been over-written, a popped return address will be read from CRSB 44. Also, when retirement 20 retires a CALL instruction, the corresponding return address is written out to a CRSB 44 entry, as will be explained.

SRSB 42 is implemented as a circular buffer having "N" entries (N being a programmable variable that can be set by a programmer or user of processor 10). SRSB 42 includes two pointers, a read pointer, STOS 46, and a write pointer, SALLOC 48. STOS 46 is used to point to an entry in SRSB 42 to read a predicted return address for the next RETURN instruction fetched. SALLOC 48 is used to points to an entry in SRSB 42 to write the next predicted return address when a CALL instruction is fetched.

Each entry included in SRSB 42 includes a pointer valid bit (V-bit) 42a, a color bit 42b, a return address field 42c, and a back pointer field 42d. Back pointer field 42d is used to hold the previous STOS 46 value, and is used to decrement ("back up") the STOS 46 pointer field when a return address is being popped from SRSB 42. V-bit 42a is written with the current value of STOS_V 50 whenever a return address is being pushed onto SRSB 42. V-bit 42a is used to indicate whether a back-pointer stored in 42d is valid, as will be explained. Color bit 42b is written with the current value of SCOLOR 52 whenever a return address is pushed onto SRSB 42. When an entry is popped (read) from SRSB 42, color bit 42b is used to determine if the return address included in that entry has been over-written, as will be explained. Return address field 42c is used to store a predicted return address that has been pushed onto buffer 40.

CRSB 44 includes multiple entries, each entry holding only a predicted return address. The total number of entries included in CRSB is programmable to correspond to a possible number of subroutine calls that may need to be stacked up due to the pipelined execution of instructions in processor 10. A single pointer, CTOS 49, is used for reading and writing CRSB 44 entries. CTOS 49 is implemented as a top of stack pointer, and is incremented when a CALL instruction is fetched, and decremented when a RETURN instruction is fetched. A CRSB 44 entry is written with a predicted return address from SRSB 42 whenever a CALL instruction is retired by retirement 20.

As described previously, SRSB 42 is a circular buffer that includes "N" entries. Buffer 40 includes a modulo N counter 54 that is used to increment the value of SALLOC 48, and also used to indicate the "wrapping" of STOS 46 above and below N. In more detail, as predicted return addresses are pushed onto SRSB 42, the value of SALLOC 48 is incremented with modulo N counter 54. When SALLOC 48 is incremented over N ("wrapping over N), the lowest bit of SALLOC 48 is set to zero to point to the first entry of SRSB 42.

Buffer 40 includes a SCOLOR indicator 47 that is a single bit field used to indicate the wrapping of STOS above and below "N". In more detail, STOS 46 pointer is up-dated to equal SALLOC 48 when a predicted return address is pushed onto SRSB. Conversely, when a return address is popped from SRSB 42, STOS 46 is up-dated to equal the back pointer address 42d (the previous STOS 46 value) stored in the entry being read. In either case, if updating STOS wraps over or under N, SCOLOR

47 bit is inverted. By inverting SCOLOR 47 each time STOS wraps above or below N, SCOLOR 47 value can be compared to a COLOR 42b value stored in a SRSB entry to determine if the return address 42c stored in the same entry has been overwritten.

Buffer 40 also includes a branch recovery structure stack (TBIT) 60 that is used to store pointers and status bits for branch recovery in a case of a mis-predicted branch path. Each entry in TBIT 60 includes a field for storing STOS 46, CTOS 49, STOS_V 50 and SCOLOR 47 whenever a branch instruction (conditional and un-conditional) is fetched by front end 16. If a mis-prediction occurs, the TBIT values are used to restore the pointers CTOS 49, STOS 46 and the indicator bits SCOLOR 47 and STOS_V 50.

A process 80 for storing and retrieving return addresses using buffer 40 is shown in FIGS. 2B-2D. Process 80 includes several separate sub-processes, 80a-80f, for performing different store or retrieve operations on buffer 40, as shown in FIG. 2A.

Sub-process 80a includes a sequence of actions, 82-92, that are performed when a CALL instruction is fetched by front end 16. Sub-process 80a includes pushing (82) STOS, STOS_V, SCOLOR and the return address being pushed by front end 16 onto a SRSB 42 entry at the SALLOC location. Sub-process 80a then sets (84) STOS equal to the current SALLOC pointer address. Sub-process 80a then determines (86) if STOS wrapped over N, and if it did, sub-process 80a inverts (88) SCOLOR 47, and also pushes the inverted SCOLOR 47 bit onto COLOR 42b field of the SRSB 42 entry at the SALLOC location. Sub-process 80a increments (90) SALLOC by one modulo-N. Sub-process 80a increments (92) CTOS.

Sub-process 80b includes a sequence of actions, 100-120, that are performed when a RETURN instruction is fetched by front end 16. Sub-process 80b includes reading (100) a return address stored in SRSB 42 at address STOS and reading (102) A return address stored in CRSB 44 at address CTOS. Sub-process 80b determines (104) if color bit 42b equals SCOLOR 47, if it does, sub-process 80b determines (108) if STOS_V 50 is set, and if it does, sub-process 80b uses (112) the return address read from SRSB 42 and sets (114) STOS_V 50 equal to V-bit 42a. If sub-process 80b determines (104) that SCOLOR 47 does not equal color bit 42b, sub-process 80b clears (106) STOS_V and uses (110) the return address read from CRSB. If sub-process 80b determines (108) that STOS_V is not set, sub-process 80b uses (110) the return address from CRSB. Sub-process 80b sets (116) STOS equal to the back pointer address 42c from SRSB and determines (118) whether STOS has wrapped under N. If STOS has wrapped under N, sub-process 80b inverts (119) SCOLOR, otherwise, sub-process 80b decrements (120) CTOS.

Sub-process 80c depicts the actions performed when a branch instruction is fetched by front end 16. Sub-process 80c includes pushing (120) the current values of STOS, STOS_V, SCOLOR and CTOS onto a TBIT 60 entry.

Sub-process 80d depicts the actions performed when a CALL instruction is retired by retirement 20. Sub-process 80d includes writing (122) the predicted return address for the retiring CALL instruction to CRSB at the CTOS address stored in TBIT for this CALL instruction.

Sub-process 80e depicts the actions performed when there is a branch mis-prediction by front end 16. Sub-process 80e includes setting (126) STOS, STOS_V, SCOLOR and CTOS to values stored in TBIT 60. Sub-process 80f also includes setting (128) SALLOC equal to STOS plus one modulo N.

Referring to FIG. 3A, a second embodiment of a two part return address buffer 40A is shown. Buffer 40A differs from buffer 40 by not including SCOLOR 47, STOS_V 50 and not including a CTOS storage location for branch recovery in TBIT 60a. Buffer 40A also differs from buffer 40 by including a SRSB/CRSB indicator 56. During operation of buffer 40a, STOS may be used to address entries in both SRSB and CRSB. SRSB/CRSB indicator 56 is used to determine which buffer, SRSB

42 or CRSB 44, is being read from or written to, as will be explained. SRSB/CRSB indicator 56 is stored with every STOS 46 stored, that is, in both the SRSB 42 back pointer field 42d and in each TBIT entry for STOS.

Referring to FIGS. 3b-3c, a process 140 is shown for storing and retrieving return addresses using buffer 40A. Process 140 includes several separate sub-processes, 140a-140f, for performing different store or retrieve operations on buffer 40A. Process 140 differs from process 80. In more detail, during the performance of process 140 using buffer 40A, CTOS pointer 49 is only incremented when a CALL instruction is retired and only decremented when a RETURN instruction is retired. Also, since STOS may be used to address either SRSB or CRSB, indicator 56 is used to determine which buffer is to be read.

Sub-process 140a includes a sequence of actions, 142-146, that are performed when a CALL instruction is fetched by front end 16. Sub-process 140a includes pushing (141) the predicted return address into return address field 42c, STOS pointer (and SRSB/CRSB indicator 56) into back pointer field 42d of the SRSB entry pointed to by SALLOC. Sub-process includes setting (142) V-bit field 42a in SRSB entry pointed to by SALLOC. Sub-process 140a includes setting (144) STOS equal to SALLOC and incrementing (146) SALLOC with modulo N counter 54.

Sub-process 140b includes a sequence of actions, 148-158, that are performed when a RETURN instruction is fetched by front end 16. Sub-process 140b includes determining (148) if STOS points to CRSB using the SRSB/CRSB indicator 56. If sub-process 140b determines (148) that STOS points to CRSB, sub-process 140b reads (156) the return address from CRSB using the STOS pointer and decrements (158) STOS pointer 46. If sub-process 140b determines STOS does not point to CRSB, sub-process 140b determines (150) if SRSB entry pointed to by STOS is valid (V-bit 42a is set). If sub-process 140b determines (150) that SRSB entry is valid, sub-process 140b reads (152) return address from SRSB at address STOS and sets (154) STOS and SRSB/CRSB indicator 56 to equal back pointer address 42d from SRSB. If sub-process 140b determines (150) that SRSB entry is not valid, sub-process 140b reads (156) return address from CRSB at address STOS and decrements (158) STOS pointer 46.

Sub-process 140c includes an action (160) that is performed when a branch instruction is fetched by front end 16. Sub-process 140c includes pushing (160) STOS (and SRSB/CRSB indicator 56) into TBIT 60.

Sub-process 140d includes a sequence of actions, 162, 164 and 166, that are performed when a CALL instruction is retired by retirement 20. Sub-process 140d pushes the predicted return address 42c for the CALL instruction into CRSB 44 at the entry pointed to by CTOS. Sub-process 140d increments (144) CTOS. Sub-process 140d clears (146) the valid bit in the SRSB 42 entry pointed to by STOS from TBIT, if the SRSB/CRSB indicator bit is pointing to SRSB. If the SRSB/CRSB indicator bit is pointing to CRSB, the valid bit is not cleared.

Sub-process 140e includes an action, 170, that is performed when a RETURN instruction is retired by retirement 20. Sub-process 140e includes decrementing (170) CTOS.

Sub-process 140f includes a sequence of actions, 172 and 174, that are performed when a branch is mis-predicted by front end 16. Sub-process 140f includes setting (172) both STOS and SRSB/CRSB indicator 56 equal to STOS and the indicator from TBIT 60. Sub-process 140f also includes setting (174) SALLOC equal to STOS plus one.

The invention is not limited to the specific embodiments described above. We mentioned using a single bit indicator for SCOLOR 47. However, more bits could be used to implement SCOLOR which would enable more "levels" of wrapping back and forth through a circular stack buffer.

Accordingly, other embodiments are within the scope of the following claims.